CajunBot-II: An Autonomous Vehicle for the DARPA Urban Challenge

Technical Paper

Submitted by

Team CajunBot

Towards requirements for the

DARPA Urban Challenge

Authors: Arun Lakhotia[♠], Padraic Edgington[♠], Suresh Golconda[♠], Anthony Maida[♠], Pablo Mejia[♦], Gunasekaran Seetharaman[♥]

Affiliations:

[♠] University of Louisiana at Lafayette, Lafayette, LA
[♥] Air Force Institute of Technology, WPAFB
[♦] C&C Technologies, Lafayette, LA

Address correspondence to: Arun Lakhotia (arun@louisiana.edu)

The project has been supported in part by funds from Louisiana Governor's Information Technology Initiative.

DISCLAIMER: *The information contained in this paper does not represent the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency (DARPA) or the Department of Defense. DARPA does not guarantee the accuracy or reliability of the information in this paper.*

 Released on 6/1/2007

CajunBot-II: Autonomous Vehicle for DARPA Urban Challenge

1. Executive Summary

CajunBot-II is a 2004 Jeep Wrangler Rubicon equipped with an INS/GPS for localization. It has is equipped with a variety of environment sensors, including SICK LMS LIDAR sensors, Alasca XT LIDAR sensors from Ibeo, stereo vision, and radars.

The software system is decomposed into a collection of programs communicating via a blackboard supported by a layer of middleware. Localization from the GPS and INS is improved by a Kalman filter based algorithm using a SICK LIDAR, and vehicle commands. Environment sensor data is fused using probabilities derived from the operating context. The mission is mapped into a series of maneuvers using the mission, vehicle state, and environmental information. A maneuver is applicable only on a path with a certain geometry. Execution of a maneuver takes into account the dynamic driving conditions. Human driving experiences are used to catalogue the maneuvers needed to satisfy the UC requirements.

The team uses a 3-D physics based simulator for virtual testing. By using an RNDF file to create a world, the simulator provides a seamless way to perform the same tests in the field and in the lab. This compresses the debugging effort because scenarios observed in the field can be recreated in the lab.

2. Introduction

Team CajunBot consists primarily of faculty and students of the University of Louisiana at Lafayette. The team also has a member affiliated with the Air Force Institute of Technology, C&C Technologies, and the US Geological Survey; each of whom contributes his individual time. Team CajunBot's vehicle CajunBot, a six-wheeled ATV, was a finalist in the 2004 and 2005 DARPA Grand Challenge events. For the 2007 Urban Challenge (UC), the team has developed CajunBot-II on a 2004 Jeep Rubicon Wrangler platform (Figure 1).

Figure 1: View of CajunBot-II showing primary sensors

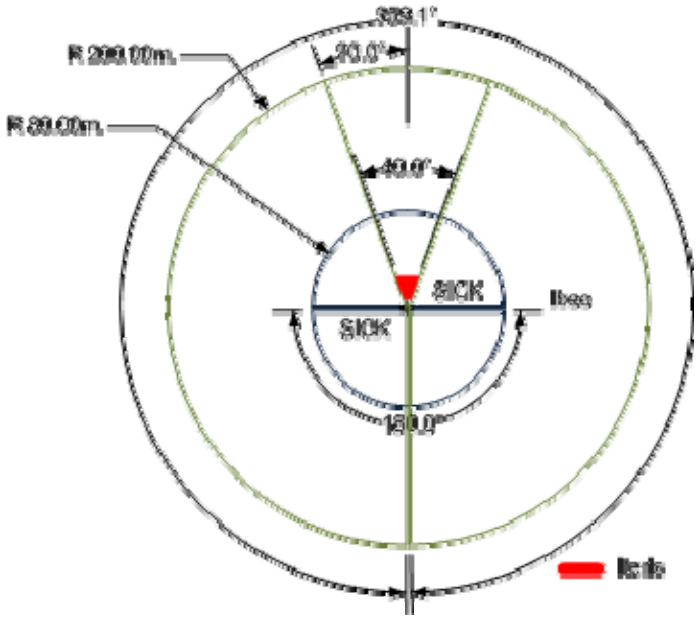

Figure 2: Sensor field of view

This document describes the technical approach used by Team CajunBot. Section 3 presents an overview of the all the hardware and the software architecture. Section 4 presents an overview of the software algorithm. Section 5 evaluates the system wrt UC requirements.

3. Overview

This section provides an overview of the entire system and, where appropriate, discusses the design choices underlying the decisions.

3.1. Hardware Overview

3.1.1. Automotive

CajunBot-II is developed on a stock 2004 Jeep Wrangler Rubicon with three significant aftermarket alterations. First, the vehicle has been fitted with an electronic drive-by-wire system from Electronic Mobility Corporation (EMC) of Baton Rouge, LA. Second, the vehicle is fitted with two high-output alternators, each with maximum current output of 250 A. Third, the vehicle's bumper and roof have been modified to provide mounting locations for sensors.

3.1.2. Sensors

CajunBot-II uses an RT3102 INS from Oxford Technology Solutions fed with a Starfire differential correction signal in RTCM format output by a C&C CNAV receiver. To bound the GPS errors, the INS also receives input from a TTL level optical wheel encoder.

The design of the environment sensor configuration has evolved through three iterations. The current configuration consists of two Alasca XT LIDAR sensors from Ibeo (Fuerstenberg, 2002), three SICK LMS LIDAR sensors, and one AutoVue Lane Departure Warning (LDW) system

from Iteris (van dan Elzen, 2004). The Ibeo LIDAR sensors are mounted on the left and right edges of the front bumper offset ± 20°. One SICK LIDAR sensor is mounted on the center of the front bumper, scanning a horizontal plane parallel to the ground. The second LIDAR unit is mounted on the rear of the vehicle, in the place where a spare tire is normally mounted. This LIDAR also scans a horizontal plane parallel to the ground. The Iteris LDW is mounted on the windshield, inside the vehicle.

Figure 2 shows the field of view of the three sets of sensors. The red region is the area viewed by Iteris LDW. It has a 40° FOV with a range of 6 m to 22 m. The LDW, originally designed to provide audible warning to the driver when the vehicle departs a lane, has been modified by Iteris to provide information about curvature of lane as well as position and orientation of the vehicle within the lane. The SICK LMS LIDAR sensors operate at 75 Hz and have a range of 80 m with 180° FOV. Since the front and rear SICK LIDAR sensors are mounted to scan parallel to the ground, the two LIDAR units effectively see all around the vehicle, except the parallel region between the sensors. Each Ibeo LIDAR sensor has a 240° FOV, of which about 10° is occluded by the vehicle. Together the two LIDAR sensors cover about 359.1° around the vehicle, with an overlapping region in the front. These LIDAR systems operate at 16 Hz and have an effective range of 200 m. Unlike a SICK LIDAR sensor that shoots a single laser beam at a time, an Ibeo LIDAR unit shoots four simultaneous beams , thus covering a volumetric area (Fuerstenberg, 2002),. An Electronic Computer Unit (ECU) manufactured by Ibeo can input the point cloud data from multiple LIDAR sensors and identify dynamic and static obstacles in the scene (Fuerstenberg, 2003),.

CajunBot-II also has the capability to process data from Eaton radars. With the acquisition of the Ibeo LIDAR system, these radars have less significant. They are retained to provide additional information in the event that the Ibeo LIDAR sensors fail.

3.1.3. Computers

CajunBot-II has three computers referred to as Main, NTP, and Logger. Each is a Single Board Computer in EPIC form factor with a 1.8 GHz Pentium M. The Main computer is used for processing SICK LIDAR data, path planning, and steering. The NTP computer provides the network time protocol service for the other computers. The Logger saves all the data acquired during a run and also has the ability to broadcast the data over a wireless network. The Main and Logger computers run Fedora Core 5 (FC5). The NTP service, although light-weight, is provided by a separate machine because the PPS patch needed for NTP is available for the Linux 2.4 kernel, and not for the Linux 2.6 kernel used in FC5. The Ibeo LIDAR system and Iteris LDW have their own computing units. All the computing devices, sensors, and other equipment communicate over the Ethernet, connected through a Gigabit Managed Switch from Dell.

3.1.4. Emergency Control

There are multiple mechanisms to take emergency control of the vehicle. As required by DARPA, the vehicle has two emergency stop buttons, one on each side of the vehicle. It has interfaces for the DARPA E-stop and also a wireless E-Stop from TORC technologies. Besides E-stop mechanisms, the vehicle also has two mechanisms to take manual control. The first mechanism consists of a console provided by EMC. This can be used by an operator inside the vehicle. The second mechanism is an RC controller, which allows one to remotely take control

of the vehicle, such as from a chase vehicle. Any of the wireless mechanisms can be deactivated. However, as a matter of additional precaution when a wireless mechanism is activated if its receiver loses communication with the transmitting device, the E-stop mode is automatically triggered and the vehicle is killed.

CajunBot-II also has an additional hardware level safety mechanism to prevent a runaway vehicle in case the main computer fails or if the control software hangs. The Labjack D2A converter, which generates signals for the EMC drive-by-wire system, contains a watchdog timer. If the converter does not receive any signals within a specified time, it is programmed to move the EMC controls to pre-designated levels, which are currently set to aligning the steering straight, releasing the throttle, and pressing the brakes.

3.1.5. Power management

The power needs of the EMC drive-by-wire control system and vehicle electronics is met by one of the high-output alternators mentioned in Section 3.1.1. The second alternator generates power for all the computers, sensors, and other devices. This alternator feeds into two Absorptive Glass Mat (AGM) batteries, which then feed to DC-DC converters, before feeding the power to the electronics. CajunBot-II also has a 100 A 13.8 V DC Rack mount power supply for providing shore power, such as in the lab.

3.2. Design Criteria for Software Architecture

The CajunBot-II software architecture satisfies the following design criteria:

Device Independence. There are multiple vendors for sensors, such as, GPS, INS, IMU, LIDAR, and so forth. The core algorithms of the system should not depend on the specific device. It should be possible to replace an existing device with another make/model or to introduce a new device while making only localized changes to the system.

Algorithm Independence. Development of the system is an iterative process, which involves choosing between competing algorithms for the same task. It should be possible to develop each algorithm in isolation, that is, in a separate program, and switch the algorithm being used by selecting some configuration values.

Scalability. The computational requirements of the system may vary as the system's design evolves. For instance, if CajunBot-II did not perform adequately with two LIDAR sensors and there was a need to add a third, it would require more computational power. It should be easy to add additional sensors and also seamlessly distribute the application on multiple computers.

Off-line Testability. The definitive way to test an autonomous vehicle is to run it in the field. However, it is infeasible to test every change in each sub-module of the system by running the vehicle in the field. The system should enable off-line (in the lab) testing of various components and compositions of components.

Ease of Debugging. To debug a system one needs access to internal data of the system. When debugging an AGV, it is most valuable if the internal data is available in real-time, when the vehicle is running. Debugging also requires performing the same operation over and over again, a process that is very expensive when done in the field. The system should support (1) real-time monitoring of internal state of its various components and also (2) the ability to replay the

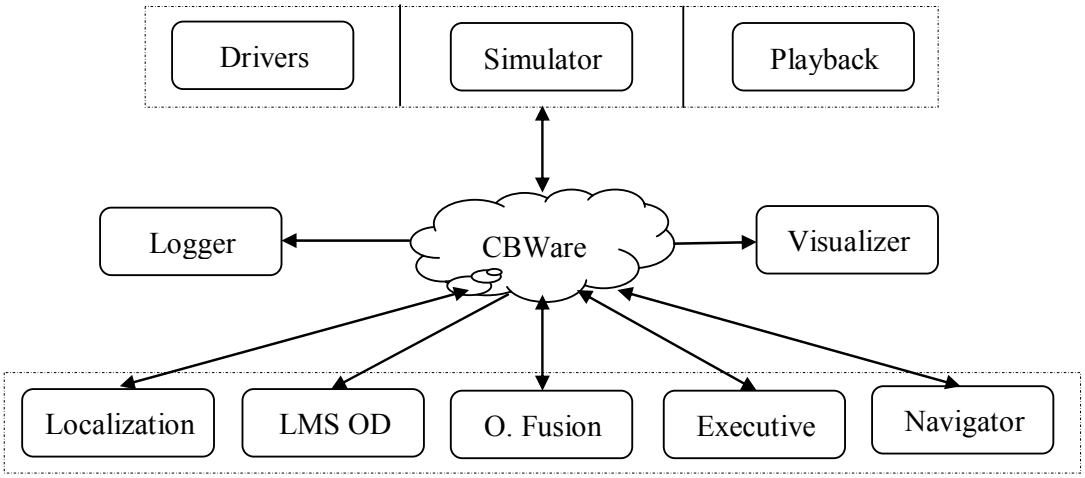

Figure 3: Architecture showing the flow of information

internal states time-synchronized. The system should also support (3) presenting the data, which is expected to be voluminous, in a graphical form to enable ease of analysis.

Composability. This criterion was not used in CajunBot, and was introduced for CajunBot-II. Since the UC requirements are inherently complex, it is obvious that the capabilities will be developed incrementally. For incremental development it is important to localize the effects of creating and removing modules. Thus, it should be possible to create a complete or partial system by composing implementations of various capabilities. Creating modules that are composable would ensure that the project continues to move forward.

3.3. Software Overview

At the highest level a system's architecture may be described by its major components and how data flows between them. We use two views to present the flow of information; Figure 3 shows the physical flow and Figure 4 shows the logical flow. The physical flow shows how the software components are distributed across computers. It also shows the mechanism for transporting data between components. The logical flow connects the producer of a data to its consumers, and hides the transport mechanism or the computing device.

The major software modules of the system are *Localization*, *Obstacle Detectors*, *Obstacle Fusion*, *Executive*, and *Navigator*. These modules implement the AGVs intelligent control. The Obstacle Detectors is a collection of programs, one program for each different type of sensor. These are discussed in the next section. All other modules are considered Support Modules, and are described in this section. The *Drivers*, *Simulator*, and *Playback* modules are mutually exclusive, Figure 3. While the Drivers module provides an interface to a physical device, the Simulator, and the Playback modules provide virtual devices, as described below. Only one of the three modules can be active at any time. The mutual exclusion of the three modules is annotated in the architecture diagram by the walls between these modules.

The physical flow of data is managed by CBWare, the CajunBot Middleware.

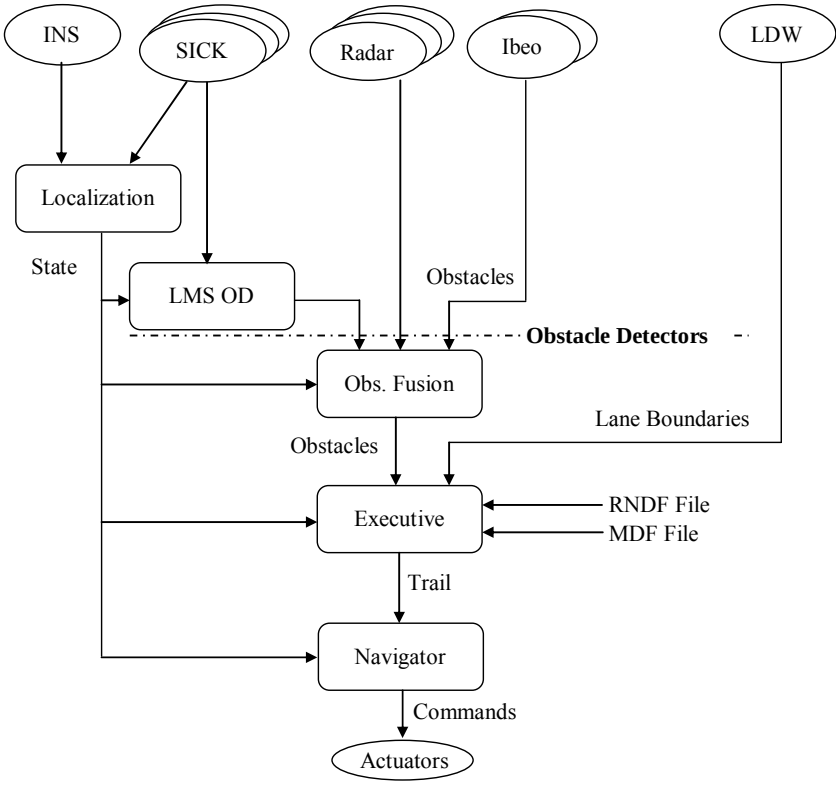

Figure 4: Architecture showing logical flow of information

3.3.1. CajunBot Middleware

CBWare (Venkitakrishnan, 2006) provides the blackboard for communication between the modules. Each module is implemented as a collection of one or more processes. The processes may be concurrent—running on the same machine—or distributed—running on different machines. CBWare provides the mechanism to transport the data between processes, whether concurrent or distributed. Except for the properties of the data written to or read from CBWare, a module in the system does not need to know anything else about the module that has generated or will consume the data. Besides containing descriptors for the immediate situation, the CBWare blackboard also contains information about the system state, the AGV's current activity context and goals, and also state information about the environment. An example of why this last category of information is needed shows up at intersections. The AGV must exhibit proper queuing behavior at an intersection. To achieve this, it must have a memory of which cars have reached the intersection after its own arrival.

CBWare serves the same purpose as NIST's Neutral Message Language (NML) (Shackleford et al., 2000), Simmons & Dale's CMU-IPC (Simmons and James, 2001), or RTI's NDDS (Pardo-Castellote and Hamilton, 1999), to cite a few middleware frameworks for real-time, distributed systems. A detailed comparison of CBWare with these systems may be found in (Venkitakrishnan, 2006).

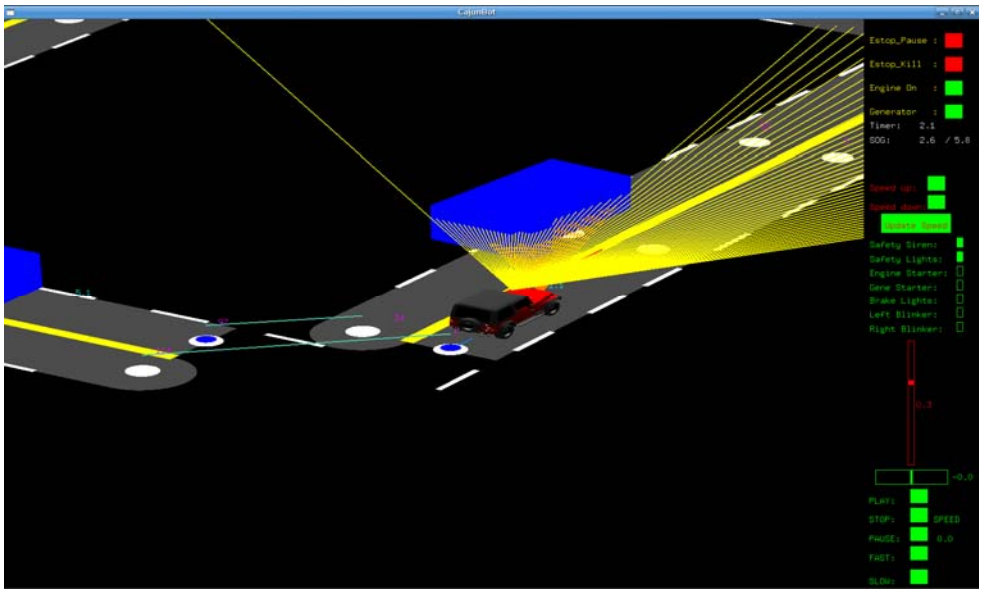

Figure 5: Simulator showing detection of moving object

3.3.2. Drivers

Device independence is achieved by having a separate program, referred to as a Driver, to interact with a particular device. The Drivers are divided into two classes. The first class is Sensor drivers, which read input data from sensors, such as the INS and LIDAR sensors. The second class is Control drivers, which control devices, such as throttle, brakes, safety lights, siren, and indicator lights.

Besides hiding the details of communicating with the device, a Driver also transforms the data to match the units and conventions used by the rest of the system. For instance, the CajunBot system measures angles in the anti-clockwise direction, with east as zero. If an IMU or INS uses any other convention for measuring angles, its corresponding device driver transforms angles from the device's convention to CajunBot's convention. Similarly, most GPS and INS equipment tends to provide vehicle position in latitude/longitude, which is translated by the driver to UTM coordinates, as used by the CajunBot system.

3.3.3. Simulator

Off-line testability is a direct outcome of device independence. Because the core algorithms have been disassociated from the other information processing modules, the data in the system does not have to interact with an actual device. The algorithms can then easily interact with virtual devices and a virtual world. The Simulator module and the Playback module—which will be discussed later—create a virtual world in which the core algorithms can be tested in the laboratory.

CajunBot's simulator, CBSim, is a physics-based simulator developed using the Open Dynamics Engine (ODE) physics engine. Along with simulating the vehicle dynamics and terrain, CBSim also simulates all the onboard sensors. It populates the same CBWare blackboard with data in the

same format as the sensor drivers. It also reads vehicle control commands from the blackboard and interprets them to have the desired effect on the simulated vehicle.

To simulate an urban environment, the simulator provides the ability to simulate traffic. Each vehicle in the simulated traffic is operated by a software agent whose behavior can be scripted externally. A simulated vehicle too occupies a 3-D space. Thus, the simulated sensors can 'see' these vehicles just as the real sensors will see vehicles in the real-world. Figure 5 shows a screenshot of visualizing the simulator. The black and red vehicle shows simulated CajunBot-II. The yellow segments denote simulated SICK LIDAR beams. The blue box is a simulated moving object, driven by a software agent.

In many respects CBSim is like Gazebo (Gerkey, 2003; Vaughan, 2000) of the Player/Stage project. Both provide the ability to simulate the robot, the sensors, and the environment. Unlike Gazebo, however, CBSim uses an RNDF file to create the environment. The virtual objects in the environment are placed wrt to the RNDF file. CBSim also provides the ability to simulate traffic on the RNDF file. These capabilities, shown in Figure 5, make it much simpler to create virtual worlds for offline testing. RNDF files created for the field can be directly used in the simulator.

CBsim also provides us with the ability to perform batch testing. We utilize this capability in two ways. First, we periodically run the entire system through a large set of tests using automatically generated MDF files for DARPA's Sample RNDF file. Such tests have helped us in identifying several special cases that we could not have found by our field tests (due to limited access to testing areas). Second, instead of merely creating nightly builds of the system, we also run some simple tests which help in finding integration errors.

3.3.4. Playback

Offline-testing and debugging is further aided by the Playback module. This module reads data logged from the disk and writes it to the blackboard. The order in which data is written is determined by the time stamp of the data. This ensures that all data is played back in the same relative order.

This simple act of playing back the logged data has several benefits. In the simplest use, the data can be visualized over and over again, to replay a scenario that may have occurred in the field or the simulator. The Playback module also offers the ability to only replay the portions of a scenario that are useful for testing. In a more significant use, the Playback module can also be used to test the core algorithms with archived data. This capability was instrumental in helping us refine and tune our obstacle detection algorithms during the 2005 GC (Puntambekar, 2006). It is our common operating procedure to drive the vehicle over some terrain—such as during the DARPA National Qualifying Event—playback the INS and LIDAR data, apply the obstacle detection algorithm on the data, and then tune the parameters to improve the obstacle detection accuracy.

3.3.5. Visualizer

Real-time and off-line viewing is supported by CBViz, the Visualizer module. CBViz is an OpenGL based program that presents a graphical view of the world as seen by the system. It accesses the data to be viewed from blackboard. Thus, CBViz may be used for live visualization

of data during field tests and simulated tests, as well as visualizing logged data using the Playback module.

Since communication between processes occurs using CBWare, CBViz can visualize data flow between processes. We have also found it beneficial to write to the blackboard data that is otherwise internal to a process. This capability has been essential to achieving the Ease of Debugging design criterion listed above.

4. Overview of algorithms

This section provides an overview of the algorithms used in the core modules: Localization, Obstacle Detectors, Obstacle Fusion, Executive, and Navigator.

4.1. Localization

In order to determine where the vehicle is and where it needs to go an AGV needs to know its location. CajunBot-II contains myriad sensors which are capable of providing information about its position at any time. While some of the sensors are designed specifically for determining the position of the AGV it makes sense to combine the information from the other sensors to create a more accurate position.

The Oxford RT3102 INS includes three angular rate sensors (gyros), three servo-grade accelerometers, GPS receivers and a connection for wheel encoder data. The RT3102 uses a Kalman filter to combine the information from the three information sources. Using the inertial sensors and differential correction signals, the RT3102 system generates position and orientation information in real-time at a fast update rate (100 Hz). The position estimate it outputs is accurate within 0.4 m accuracy under dynamic conditions and the heading accuracy is 0.1°.

If the GPS signal were available and accurate all the time, then a separate localization algorithm would not be necessary. Unfortunately, the GPS receivers have trouble when their view of the sky is impeded, or there are buildings nearby to produce echoes. Since trees and buildings are common in most urban environments, we cannot expect the GPS to always be available, or even much at all. The RT3102 incorporates an IMU sensor and wheel encoder to compensate for short GPS outages and improves upon the standard GPS solution when the GPS is available. However, the IMU information is relative to a previous state and any error present in a solution is propagated and usually magnified over time. The wheel encoder data can help to mitigate this problem, but it only provides information about the distance traveled along the main axis of motion.

To provide a more complete solution to the problem we expand upon the basic Kalman filter premise. Localization algorithms of this type are common, but tend to be limited to a single prediction, single sensor type. The Navigator module, discussed later, provides the information needed for a prediction. While the information processed by the RT3102 system is not available separately it can still be combined with the information available from the SICK LIDAR sensor to create a more accurate solution.

CajunBot-II contains three different types of sensors when examined from a localization point of view. The first, and simplest, is the global type of sensor. In this case the GPS gives a position which is irrespective of anything in or around the vehicle. This type of sensor is very useful for containing the amount of error present in the localization of a robot, but can easily be disrupted.

```
Localization (p_position, command,
                ins, lidar, p_lidar)
{
   lidar_transformation = AlignLidar (lidar, p_lidar);
   while (uncertain)
   {
      y = Select (p_position);
      x = Predict (a, command);
      if (gps)
         w = CompareGps (x, ins);
      else
         w = CompareIns (x, y, ins, p_movement);
      w += CompareLidar (x, y, lidar_transformation);
      position += {x, w};
      uncertain = !Enough (position);
   }
   return Choose (position);
}
```

```
AlignLidar (lidar, p_lidar)
{
   while (error > threshold)
   {
      foreach point in lidar
      {
         find closest point x;
         find matching range point y;
      }
      compute least-squares solution <ω1, T1> for all x
      compute least-squares solution <ω2, T2 > for all y
      Update (lidar, <ω2, T1> );
      Update (solution, <ω2, T1>);
      error = CalculateError (lidar, p_lidar);
   }
   return solution;
}
```

Figure 6: Pseudocode for Localization (Fox, 2003) and AlignLidar (Lu, 1997)

The second type of sensor produces relative information about some aspect of vehicle motion. IMU sensors fall into this category. These work well since they will always provide some information about the vehicle's movement, but they will only sense information relative to a previous state. In the case of IMU sensors, they provide information about accelerations and thus have two places where error will accumulate: once when determining the velocities of the vehicle and once when determining the absolute position. The third type of sensor observes something about the local environment or the vehicle's interaction with the external world and uses this information to determine the robot's movement. This category includes cameras, radar, sonar, LIDAR, wheel encoders and many others. These sensors generally suffer fewer problems than global sensors from occlusion, but instead accumulate error in position. This puts them between global and relative sensors in terms of strengths and weaknesses.

While using more information to make a decision generally leads to a better decision, by combining the information from these three types of sensors the problems inherent in using any one type can largely be canceled out. In order to make our solution as accurate as possible, the type of information being obtained from the sensors has to be kept in mind when processing it. When the GPS is working the RT3102 system generates a prediction that has the properties of a highly accurate global sensor. However, when the GPS information is not available to the INS system the resulting calculations must be treated as a largely relative source.

The main Localization algorithm as described in Figure 6 implements a modified form of the KLD particle filter (Fox, 2003). A previous particle is selected proportional to its weight. That particle is then modified based on the expected effects of the commands given by the Navigator module. That prediction is then compared to the output of the INS system. The method of comparison is based on the availability of GPS data for the time step. The prediction is then compared against the LIDAR data to obtain a final weight for the new particle. The system then decides if it has created enough particles to make a decision for this time step.

The CompareGps function uses the standard Gaussian probability formula to determine the likelihood of the given sample. The CompareIns function uses the same theory, but compares the movement between the sample and the prediction to the expected value. The expected value

is calculated by using the movement of the previous cycle combined with the acceleration data provided by the IMU sensors.

The comparing a position with a LIDAR scan is more complicated since the LIDAR sensors do not directly indicate anything about the position or movement of the vehicle. Instead the LIDAR sensors tell the system about the relative location of other objects in the world. To turn this information into knowledge about the AGV's position, two successive range scans are compared to determine the transformation which when applied to the first range scan will produce the second. For this part we utilize the algorithm described by (Lu, 1997) for aligning range scans. Once the range scans are aligned, it is simple to compare the predicted movement to the results from the range scan alignment.

The pseudocode for AlignLidar in Figure 6 details how to determine the movement that occurred between two LIDAR scans. The algorithm is iterative, so it is repeated until the result is considered sufficiently accurate for use. At each iteration, two different algorithms are applied to determine the transformation that occurred between them. The first is a basic closest-point algorithm which picks the nearest point to associate a given point to. This method prefers to vary the translation of the scan to make them overlap. The second method tries to select a point that is within the same range and prefers to vary the angle of rotation. The least-squares solution is calculated for each algorithm separately. To determine the final solution for the iteration, the translational component is taken from the closest-point algorithm, and the rotational component is taken from the matching-range algorithm. This is then compounded with the previous solutions and a new error is calculated.

4.2. Obstacle Detectors

There are a variety of sensors available to detect dynamic and static obstacles. We have experimented with SICK LMS LIDAR sensors, Ibeo Ithaca XT LIDAR sensors, stereo vision cameras, and radars. The algorithm used to analyze a sensor's data to extract obstacle information is intrinsically tied to the type of sensor being used. Further, there are often multiple algorithms used for the same sensor.

Manufacturers of some sensors, such as the Ibeo LIDAR sensors (Fuerstenberg, 2003) and radars, provide components that process the raw sensor data and provide obstacle information. For such a sensor, the sensor driver is the obstacle detection program. There are also sensors, such as the SICK LMS LIDAR sensor or stereo vision camera, for which there are no readily available programs, whether from a manufacturer or from a third party, for extracting obstacle information from the sensor data. For such sensors we need to write our own obstacle detection programs.

We have developed algorithms for obstacle detection using SICK LMS LIDAR units (Puntambekar, 2006). The algorithm we used in the 2005 GC was very robust, producing no false obstacles even though CajunBot, the six-wheeled ATV, had no suspension and hence no stabilization of sensors. While our algorithm produces very reliable results, it is not adequate for the UC for two reasons. First, the 80 m range of the SICK LMS sensors is inadequate for meeting the UC requirement of pulling into a 10 s gap in traffic when vehicles are traveling at 13.4 m/s (30 mi/hr).

To fulfill this requirement we are employing two Ibeo LIDAR sensors in a single array (Fuerstenberg, 2002). The Ibeo LIDAR sensors have a range of 200 m so they are easily able to

cover the 134 m range needed for the traffic merging requirements. They are laid out in such a way that they cover the majority of the area around CajunBot-II (Figure 2). However, the configuration leaves a small blind spot directly in front of the vehicle and a continuous blind spot behind the vehicle. The front area is covered by a single SICK LIDAR sensor. Because the rear region does not require long range visibility it can also be covered by a SICK LIDAR sensor.

In order to integrate well with the Ibeo sensors we implemented an algorithm similar to the one used by the Ibeo processing system to analyze the SICK LIDAR data. This algorithm uses a basic Kalman filter to predict the position of an object based on its previous actions and melds that information with the information observed by the SICK LIDAR sensors.

Before any of the information acquired from the sensors can be used for higher level functions it is transformed into a common set of data structures. This design also helps to mitigate the cost of utilizing a myriad of different sensor types, allowing us to experiment with new technologies and algorithms as they become available.

4.3. Obstacle Fusion

In order to make decisions about the world, a system needs to have a unified information system that details everything that the system needs to know about its environment. For the UC, one of the most important things for the AGV to know about its environment is the presence of and expectations about objects in the world.

Because we have the ability to utilize multiple sensors to detect obstacles; the information from each sensor must be synthesized to create a single belief about the actual state of the world outside. This Obstacle Fusion module also helps to meet the 'sensor independence' design criteria. The main objective for the Obstacle Fusion module is to allow the Executive module to be independent of any effects of a sensor type.

The most basic method for fusing the obstacle information is to simply take all the obstacles detected by the various systems and combine them into one big conglomeration. This method can work effectively when extremely high quality sensors are available, or all possible processing on the sensor information has been done before the obstacles are fused. Unfortunately, all sensors have some error with their results and to perform every possible mechanism to clean up the sensor data is far from feasible.

To improve on the basic method, we could employ a daisy chain algorithm which only believes the most accurate sensor unless that sensor becomes faulty. The daisy chain method keeps the Executive module from being plagued with false returns, but creates gaps in the places where a less reliable sensor is more accurate.

Regrettably, all sensors suffer from some type of error. Cameras are susceptible to changes in lighting and light refraction. LIDAR sensors have trouble detecting black objects. Radar sensors often miss objects that are not moving relative to the sensor. Virtually all sensor systems have problems with a reflected signal of some form. Fortunately, the majority of weaknesses are limited to a single type of sensor. This means that by employing the information from multiple sensors dynamically it is possible to compensate for the weaknesses of any one sensor type. Thus we can use radar to detect black cars on the road, or a LIDAR sensor to find objects in partial shade.

To perform the Obstacle Fusion we use a probabilistic method that is sensitive to the dynamic changes in the world and is knowledgeable about the various sensor types that are providing information. A basic example of this type of algorithm would simply combine the obstacles and weight them based on the accuracy of the sensor. This allows objects seen by multiple sensors to be rated very highly, and objects seen by only a single sensor to be rated as unlikely. We expand on this method to work dynamically with the responses given by the various sensors. An example of how this works is in the case of combining LIDAR and radar information: A LIDAR sensor rarely produces spurious results, but in the case of a dark or clean car the laser beam can be lost. So, if we want to combine the radar data with the LIDAR data we can decide to only give a high weight to a mismatched radar signal if the LIDAR beam does not have a return.

While any set of rules about the sensor system interactions will have problems in certain areas, the set can be made sufficiently complex to compensate for most of the inconsistencies between sensors leaving a highly accurate amalgamation of obstacles.

4.4. Executive

The Executive module is responsible for the intelligent control of the vehicle. It provides the control, sequencing, and deliberative aspects of the classical 3-layer architecture (Gat, 1998). But the specific method of decomposition used is different. In our architecture, the first layer provides the capability for the vehicle to travel without any obstacles. The second layer consists of augmenting the first layer to introduce the ability to handle static obstacles. The third layer, again developed by augmenting the second layer, introduces the ability to perform in traffic and to respond to unexpected scenarios.

The Executive module generates a *Trail*, a sequence of waypoints annotated with speed, to guide the vehicle towards its mission. The actual task of following the Trail is performed by the Navigator module. The Executive module reads the following from the blackboard: vehicle's state from localization, the obstacle information obstacle fusion, the lane information from LDW, the RNDF, and the MDF file. It writes the Trail to the blackboard for the Navigator to follow.

The pseudocode of Figure 7 and Figure 8 describe the underlying logic. In UC terminology a mission, given in an MDF file, is a sequence of checkpoints. The AGV must visit each checkpoint in the specified order. To compute the Trail the algorithm computes a *Path* and a *Maneuver plan*. A path is largely a sequence of RNDF waypoints. A maneuver plan is a sequence of instantiated maneuvers. A maneuver is a tactical operation that advances the vehicle in a specific situation. The maneuvers needed to meet the UC requirements are: *follow the lane*, *change lane*, *pass vehicle*, *navigate intersection*, *three-point turn*, *navigate in free zone*, *pull-in-parking spot*, *pull-out-of-parking spot*, and *wait*.

```
ExecuteMission (rndf, mission)
{
   construct rndf_graph from rndf
   repeat
   {
       UpdateRndf (rndf_graph);
       path = FastestPath (rndf_graph, mission);
       maneuver_plan = PlanManeuvers (path);
       ResetRndf (rndf_graph);
   } until (ExecuteManeuvers (maneuver_plan))
}
```

```
UpdateRndf (rndf_graph)
{
  mark edges blocked by static obstacles;
  foreach (node in rndf_graph)
    if all outgoing edges of node are blocked
       by static obstacles then
    {
        // add temporary edges
       add a 'pass-vehicle' edge, if feasible;
       add a 'u-turn' edge, if feasible;
       add a 'reverse' edge, if feasible;
    }
  }
```

Figure 7: Pseudocode of ExecuteMission and PlanPath

The first step in executing a mission, see *ExecuteMission* of Figure 7, is to construct an RNDF graph from the RNDF. Each waypoint in the RNDF becomes a node in the RNDF graph. Five types of edges are added in the RNDF graph during initial construction. These are *same-lane* edge, *change-lane* edge, *intersection* edge, *free-travel* edge, and *parking* edge. A same-lane edge is added from a waypoint in a lane to the next waypoint in the same lane. A change-lane edge is added from a waypoint in a lane to the next waypoint in another lane, where both lanes are traveling in the same direction. An intersection edge is added from an exit waypoint to its corresponding entry waypoints. The free-travel edge and parking edge represent navigation in the free zone. For navigation on the road when there may be more than one type of edge between a pair of nodes, the edge with the highest precedence is added, where an intersection edge has the highest precedence, and is followed by change-lane edge, which is followed by same-lane edge.

The RNDF graph and the static obstacles are then used to update the rndf_graph, see *UpdateRndf* of Figure 7. Static obstacles are objects that are not expected to move. In contrast, a dynamic obstacle is one that *may* move. A vehicle stopped at the intersection is a dynamic obstacle since it may move. When updating RNDF graph the dynamic obstacles are ignored. The presence of static obstacles on the route is accounted in two ways. First, each RNDF edge that traverses through an obstacle is marked as blocked. Second, three types of (temporary) edges, *pass-vehicle*, *u-turn*, and *reverse* are added to represent alternate edges to the blocked edges. These edges are temporary in that they are removed by ResetRndf after each iteration of ExecuteMission. A temporary edge is added only if all out-going edges from a node are blocked by static obstacles. The pass-vehicle edge represents going around the obstacle by traveling on the oncoming lane to pass a vehicle. The u-turn edge represents making a u-turn. The reverse edge represents simply driving the vehicle in reverse. These edges are also added only if the paths they traverse are free of static obstacles. It may not always be optimal for these alternate edges to be between RNDF waypoints. Thus, adding an edge in the RNDF graph may also require adding new nodes. For instance, when a same-lane edge is blocked, it may be replaced by a pass-vehicle edge. In some situations it may not be possible to satisfy UC constraints on passing a vehicle by adding a pass-vehicle edge between RNDF waypoints; requiring introduction of new waypoints. These details are not pertinent to understand the overall algorithm.

```
PlanManeuvers (path)
{
  // ManeuverList is a list of all maneuvers
  maneuver_plan = empty;
  while (path != empty)
   {
     found = false;
      // find a maneuver that can be applied
     for (maneuver in ManeuverList)
          if (maneuver.StaticCondition (path))
           {
             found = true;
             break;
           }
     // one maneuver must apply
     assert (found);
     // instantiate maneuver for path
     mi = maneuver.Instance (path);
     path = mi.ConsumePath (path);
     put mi in maneuver_plan;
   }
    return maneuver_plan;
}
```

```
ExecuteManeuvers (maneuver_plan)
{
   while (maneuver_plan != empty)
   {
      mi = first in maneuver_plan;
      if (mi.completed) remove mi from maneuver_plan;

      // are (new) static obstacles blocking trail
      for (mi in maneuver_plan)
       {
          if mi.trail is blocked by static_obstacles then
             // cannot execute plan
            return false;
          else
             // account for dynamic obstacles
            mi.UpdateTrailSpeed();
       }
       // progress the maneuver
       publish entire trail to the blackboard;
    }
   // plan executed
   return true;
}
```

Figure 8: Pseudocode of PlanManeuvers and ExecuteManeuvers

The updated RNDF is then used to compute the Path—the fastest path from the current location of the vehicle to the mission. To compute the fastest path each original RNDF edge is annotated with the time estimated to travel the distance between its end-points at the speed designated in the MDF, except that the edges blocked by static obstacles are given infinite time. The time associated to the temporary edges—pass-vehicle, u-turn, and reverse—takes into account the need to stop the vehicle, to wait for traffic, and to complete the maneuver.

After computing the Path, function PlanManeuvers of Figure 8 is used to compute the Maneuver Plan. This plan is computed strictly using the Path. The function uses a ManeuverList, which is the list of all maneuvers. The *for*-loop in the pseudo-code finds a maneuver that can be applied on the Path. Associated to each maneuver is a *StaticCondition*, a condition on the Path that must exist for the maneuver to be applied. For example, the change-lane maneuver may only be applied between nodes connected by a change-lane edge. Or the intersection maneuver may be applied when there is an exit node followed by an entry node in the path. Once a maneuver is selected, it is instantiated with the specific properties of the path and placed in the Maneuver Plan. The path consumed by the maneuver is removed and the process repeated until the path is empty. The collection of maneuvers is such that for any Path generated by PlanPath the complete path can be consumed. This is tautologically achieved by the wait maneuver which is always applicable and which consumes the entire path. The wait maneuver is the last to be tested in the list of maneuvers.

The final step of executing a mission is to execute the maneuvers. This is done by function ExecuteManeuvers of Figure 8. It is possible that the vehicle has learned about more static obstacles in the environment since the time the maneuver plan was created. If these obstacles may interfere with the execution of any maneuver in the plan, the function ExecuteManeuvers fails. This failure triggers the repeat-until loop in ExecuteMission to be repeated, thus creating a

new Path, a new Maneuver plan, etc. On the other hand if a Maneuver Plan can be executed, the function UpdateTrailSpeed of the selected maneuver is called. This function sets the speed by taking into account the dynamic obstacles in the environment. The method for setting the speed will depend on the maneuver. For example, if a vehicle is observed to be traveling in front then at a certain point on the trail the follow-the-lane maneuver sets the speed to be the minimum of the RNDF speed and the speed of the vehicle in front. On the other hand, the navigate intersection maneuver, when making a left turn, sets the speed to zero if there is a moving vehicle whose trajectory may intersect with its trail. When there is no such vehicle and all other traffic rules have been satisfied, the trail speed is set to the max speed permitted. When the intersection is filled with moving vehicles, after waiting for sometime, the navigate intersection maneuver may set a very slow speed (leading to the Taxi Cab algorithm). Finally, when ExecuteMission finds that all the maneuvers can be performed, it publishes to the blackboard the concatenated trail of all the maneuvers. At the beginning of the loop, if a maneuver is completed it is removed from the list.

There are two more issues to address. a) How does the vehicle stay within the same lane when the curve represented by a sequence of waypoints does not represent the curve of the lane? How does the vehicle pass a static object in a free zone or on a segment with only one lane?

The issue of staying within the lane is addressed by how a trail is computed. Each Maneuver is defined by a sequence of waypoints. Only the last waypoint is explicitly put on the trail. The intermediate waypoints are used to compute a curve representing a RNDF lane. In addition, there is also the 'observed' lane as reported by Iteris LDW (van dan Elzen, 2004). The two lanes are fused to create a single lane. Higher precedence is given to the observed lane. When observed lane is not available, the RNDF lane is used. Discontinuities due to switching between different sources of lanes are smoothened by interpolation. A maneuver planner uses the fused lane as the trail.

The issue of passing a static obstacle in a free zone or in a single lane segment is also addressed by manipulating the trail. The corresponding maneuver manipulates the trail, moving it left or right, such that it no longer intersects with the obstacle (Trepagnier, 2006). The trail is then smoothened such that the path can be navigated by the vehicle. In general, this method of tweaking the trail is also used when the maneuver plan returned by PlanManeuvers contains only the wait maneuver. When the vehicle is traveling on a tweaked trail its speed is set very low, based on the distance to obstacles around it.

4.5. Navigator

The Navigator module takes as input the Trail computed by the Executive module. It is responsible for driving the vehicle along the sequence of waypoints in the Trail, keeping the vehicle as close to the given path as possible, while also maintaining the speed specified in the Trail. The Navigator uses a lead-lag compensator controller for controlling the vehicle's steering (Hingwe, 1998) and a PD controller for controlling its throttle and brakes.

Figure 9 illustrates the parameters influencing the steering controller. The steering controller projects a *steering-target point*[1], s, in front of the vehicle and computes the *steering position*, δ,

[1] It is called look-ahead point in the literature on steering controller (Hingwe, 1998), a term that leads to ambiguity in the context of obstacle detection.

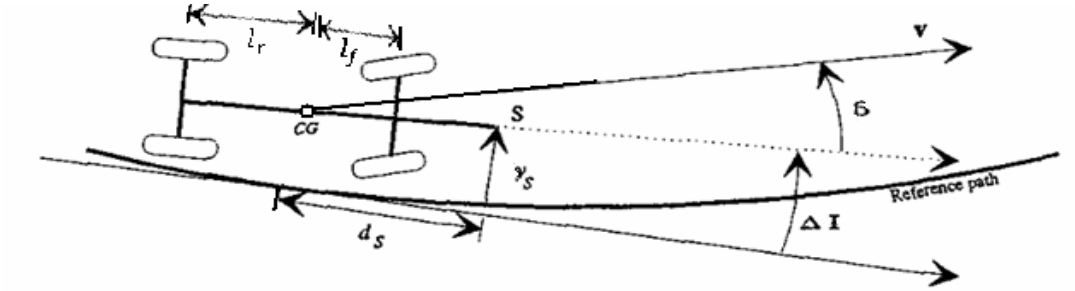

Figure 9 Illustration of parameters of the steering controller (Guldner, 1997)

that would minimize the *lateral displacement*, y, of the steering-target point from the desired path. The distance from the vehicle's center of gravity to the steering-target point is called the *steering-target distance*, d_s. The transfer function from steering angle to lateral acceleration of the steering-target point is given as (Hingwe, 1998):

$$\frac{y(s)}{\delta(s)} = \frac{\mu C_f v^2\left(M l_f d_s + I\right)s^2 + \mu^2 C_f C_r L v\left(d_s + l_r\right)s + \mu^2 C_f C_r L v^2}{(IMv^2)s^2 + \mu v\left(I\left(C_r + C_f\right) + M\left(C_f l_f^2 + C_r l_r^2\right)\right)s + \mu M v^2\left(C_r l_r - C_f l_f\right) + \mu C_f C_r L^2}$$

where:

δ = Steering angle

y_s = Lateral displacement of the virtual point

μ = Road adhesion factor (from 1 to 0)

C_f = Cornering stiffness of the front tires

C_r = Cornering stiffness of the rear tires

l_f = Length from the front axle to the center of gravity of the vehicle

l_r = Length from the rear axle to the center of gravity of the vehicle

$L = l_f + l_r$

v = Linear velocity of vehicle (speed)

M = Mass of the vehicle

I = Instantaneous yaw rate of the vehicle

d_s = Steering-target distance

These parameters are illustrated in Figure 9.

Applying the Laplacian integrator 1/s to the transfer function twice, we arrive at the transfer function between steering angle and lateral displacement of the virtual sensor. Using this transfer

function and a unity feedback (assuming the sensor makes little or no changes to the measurement) the control law can be derived. Using the root locus approach, a lead-lag compensator can be designed.

One drawback to this method is that during derivation of the control law, it does not provide a direct method to compensate for delays in the system. This is unfortunate as a .24 s transport delay is present within the steering actuation of CajunBot-II. However, when added to the system, a delay only serves to decrease settling time and oscillation to a small extent and does not affect its overall stability.

Three significant modifications were made to the control law in the actual implementation, as elaborated below.

First, the gains of each term as established in the mathematical derivation are inaccurate due to the inaccuracies of the constant values used in the original equations, as well as delays in the system, and other characteristics caused by the digital implementation of the controller. Because of this, the gains are instead shown as variables and used for tuning. In our derived control law tuning becomes very easy because the control law approximates a PD controller.

Second, the steering-target distance is computed dynamically as a function of the vehicle's speed. What may be counter-intuitive, this distance increases at slower speeds. A speed plan generates slower speed when the path has a greater lateral acceleration. The longer steering-target distance compensates for the steering delay of 0.24 s, thus preparing for the vehicle to turn ahead of time. On the other hand, at higher speeds a small change in the steering angle leads to larger change in the lateral displacement (for the same time interval). That is, the steering is more sensitive at higher speeds. A smaller steering-target distance at higher speeds ensures that the system generates gentle changes.

Third, the method for computing y_s, the lateral displacement, of the steering-target point was modified. Figure 9 shows y_s, computed as the distance from the steering-target point to the path, measured perpendicular to the vehicle's heading. When the vehicle is oriented near perpendicular to the path, such as at an intersection, the lateral displacement computed would be infinite, leading to an unpredictable behavior. While the Executive module generates smooth paths through the intersection, for reliability and to achieve algorithm independence it is important that the steering controller not misbehave if the Executive makes an error. One solution to this problem is to use a line perpendicular to the path segment to calculate the virtual sensor's lateral displacement. However, this too becomes troublesome as this can have multiple solutions and its computation is more complex. We use the following method for determining the lateral displacement. The steering-target point is determined by the normal method; however a second point, *vehicle track point*, is determined. This point is set by tracing the length of the steering-target distance along the vehicle's projected path. An example would be: if the steering-target distance is 3 m, then the steering-target point is 3 m in front of the vehicle, and the vehicle track point is 3 m along the path from the vehicle's projected position on the path. The distance between the vehicle-target point and the vehicle-track point is taken as the lateral displacement. For small heading errors, the difference in vehicle's heading and the path orientation, this computation yields values close to those from the previous method. However, when the heading error is large the new computation yields a stable solution. This method is also immune to variations in the path shape.

5. Evaluation

The performance evaluation of the system is presented along two dimensions: analytic and quantitative. The analytic evaluation verifies whether all the UC requirements are addressed by the system. The quantitative evaluation presents pertinent data gathered from observing the vehicle perform in the field.

5.1. Analytical Evaluation

Let us consider the Base Capability to be the ability of the vehicle to complete a full mission following the necessary rules, when there is no other vehicle on the road, there is no blocked road, the RNDF has sufficiently dense waypoints, the waypoints are accurate, and there is no GPS error.

It is straightforward to see that CajunBot-II has the Base Capability using only the INS, the Executive, and the Navigator. The INS will provide accurate localization information. Since there are no obstacles, the path generated by the Executive will simply be the fastest path on the RNDF graph. The eight maneuvers of the Executive address all the capabilities needed: traveling on a lane, changing lane, traveling through intersections, stopping at stop sign, turning into appropriate lane, traveling in free zone, making a U-turn, pulling in parking, and pulling out of parking. Since the waypoints are dense and accurate, the trail generated by each maneuver will be within the lane. Finally, the Navigator is capable of following a trail and significant speeds.

Having argued that CajunBot-II has the Base Capability, one can now evaluate the remaining capabilities required for the requirements in the UC Rules[2]: Basic Navigation, Basic Traffic, Advanced Navigation, and Advanced Traffic. Each of these requirements builds upon the previous one by introducing additional complexity. The complexity introduced by each requirement and how they are addressed by our system is described below.

The Basic Navigation Requirement contains the ability to negotiate static obstacles on the road. The Ibeo LIDAR system provides the capability to detect obstacles around the vehicle and the rear SICK LIDAR unit covers the area immediately behind the vehicle. At 13.5 m/s (30 mi/hr) CajunBot-II can come to a smooth stop within 20 m. The Ibeo LIDAR sensor's 200 m range is more than more than five times the range needed for the vehicle to stop 8 m away from the vehicle. The Executive module has the ability to compute path when a lane is blocked. As per the Rule the obstacles are not expected in the intersections. Thus, only the pass-vehicle maneuver needs create the correct trail to satisfy this requirement. However, if for some reason the vehicle stops too close to the obstacle, the reverse maneuver provides a way to create a clearance. The rear LIDAR sensor provides the ability to look at the back when reversing. Thus, our system has the ability to address the Basic Navigation Requirement.

The Basic Traffic Requirement includes the ability to follow a moving vehicle, to observe queuing behavior at an intersection, and to follow intersection precedence. The Ibeo LIDAR system provides the velocities of moving objects in its FOV. The velocities are provided along two axes. These sensors also provide the ability to track objects. Thus, the system has the ability to sense the conditions needed to satisfy the Basic Traffic Requirement. The necessary functionality can be introduced by appropriate encoding of UpdateTrailSpeed of the follow-the-

[2] Available on www.darpa.mil/grandchallenge

lane maneuver and the intersection maneuver. By controlling the speed of the trail these two maneuvers can satisfy the Basic Traffic Requirement.

The Advanced Navigation Requirements adds the following capabilities: travel in a free zone with static obstacles, dynamic re-planning when the roads are blocked, following road with sparse waypoints, and dealing with intermittent loss of GPS. Mechanisms to implement these capabilities are traced below.

Due to the absence of lanes in a free zone, passing an obstacle there is different from a similar operation on the road. The path to be followed cannot be found by searching the RNDF graph. The concept of a trail and the ability of a maneuver to update the trail in each iteration provides the structure needed to introduce the necessary algorithm. The algorithm used by (Trepagnier , 2006) may be used for twaking the trail to pass an obstacle. The Ibeo LIDAR sensors' 200 m radius range covers a large enough area, and sufficient information to alter the trail to pass obstacles much before coming close to the vehicles. To further prevent any mishaph, the vehicle will travel very slow through a free zone.

Dynamic re-planning does not require anything more in the Executive. The necessary capabilities follow directly from how the Executive recomputes the path when a trail thought to be free is found to be blocked. The replanning algorithm is greedy, it replans the path as soon as the sensors detect that the road is blocked. Thus, if while planning to turn into an intersection the sensors detect that the road is blocked 50 m down, the vehicle will replan a path and alter its plan of turning into the intersection.

Sparse waypoints are expected when following the lane and are handled by the follow-the-lane maneuver. This maneuver uses the lane curvature information provided by the Iteris LDW, when available, to generate a trail. It uses the waypoints in the path computed by the Executive only when no lane information is available. This same capability also enables the vehicle to navigate when the waypoints in the RNDF are inaccurate.

Method used for addressing GPS outage is addressed in Section 4.1.

Thus, our system has the ability to address the Advanced Navigation Requirement.

The Advanced Traffic Requirement adds the capabilities to deal with traffic in the free zones and to deal with traffic at intersections that do not have four-way stop. It also adds the requirements for dealing with exceptional situations such as traffic jam and erratic behavior of other vehicles. Once again the necessary functionality to deal with traffic in both the scenarios can be introduced by appropriate encoding of UpdateTrailSpeed by intersection maneuver and the navigate-through-free-zone maneuvers. Exceptional situations due to moving vehicles, whether in traffic jam or other situations, too are handled by appropriate encoding of UpdateTrailSpeed. This describes our mechanisms to address the Advanced Traffic Requirement.

Thus, our sensors provide the information needs of all the UC requirements. And our software design provides the structure needed to address all the UC requirements.

5.2. Quantitative Evaluation

The system is still under development. Implementation of the Base Capability was completed in March, 2007; the capabilities for Basic Traffic completed in May, 2007; and the capabilities for Advanced Traffic are nearing completion. The results of system and field tests of these

capabilities were tracked and are discussed below. The extensive unit and integration tests performed during the development were not tracked, and are not reported.

Table 1 Summary of system and field tests

	Runs (#)				Distance (km)		Speed (m/s)	
	Total	**< 0.5 km**	**> 0.5 km**	**> 1 km**	**Total**	**Longest**	**Median**	**Max**
Base								
Cajun Field	20	5	15	11	46.60	9.65	3.64	10.91
EVOC	31	9	22	11	55.20	31.80	7.42	17.62
Basic Traffic								
Ev. Downs	103	55	48	22	98.08	17.78	4.43	14.51
Total	154	69	85	44	199.88			

As the discussion above showed, the Base Capability forms the foundation of the system. Once the Base Capability is developed other capabilities may be introduced by making local changes to individual maneuvers. Thus, our system tests on completion of the Base Capability have been very thorough. The tests were performed in two parts. First, the system was tested in the Cajun Field parking lot of UL Lafayette. Second, a field test was performed in the Emergency Vehicle Operation Center (EVOC) of Louisiana State Police in Zachary, LA. This facility has a network of roads simulating all types of roads—from dirt roads, to neighborhood roads, to highways. Upon completion of the Basic Traffic capabilities the system tests were performed in the parking lot of the Evangeline Downs race course in Carencro, LA.

Table 1 provides a summary of the system and field tests. The 20 test runs in Cajun Field accumulated a distance of 46.6km, with a majority of the runs greater than ½ mile. The longest run was 9.65 km (6 mi). The median of the average speed for these runs was 3.64 m/s (8.14 mph); whereas the maximum speed reached by the vehicle in these runs 10.91 m/s (24.4mph). The successful completion of a 6 mi was set as a precondition for proceeding for the field test in EVOC. The field test in EVOC accumulated 55.20 km over 31 runs, with the longest run being 31.80 km (19.75 mi). During the field test the median of the average speed of the runs was 7.42 m/s (16.5 mph). The absolute (not average) maximum speed reached by the vehicle in EVOC was 17.62 m/s (39.42 mph).

The system and field tests data of the Base Capability show that CajunBot-II has the endurance necessary for being a strong contender in the UC. It can steer well, and can plan and control speed over long distances. The vehicle can attain the maximum speed recommended for UC

A significantly more number of system tests were performed after the completion of the Basic Traffic capabilities, though a majority of them were under ½ km long. Each test was performed to evaluate the ability of the vehicle to detect and avoid obstacles placed at specific locations in the RNDF. Due to the effort needed to tear down and setup the obstacles, the tests were of short length. The endurance run during this tests was 17.78 km and the maximum speed attained by the vehicle was 14.51 m/s (32.46 mph).

6. Summary

CajunBot-II is well on its way to become a strong contender in the UC. The vehicle has the electronics and sensors needed to address all the requirements. It has the power generation capacity necessary to run the computers and electronics for an extended period of time. Its

software system has the ability to utilize multiple environment sensors, including, LIDARs, radars, and cameras. Data from individual sensors is analyzed independently, and then fused. The fusion algorithm takes into account the reliability of each sensor in a particular context. The planning component of the software system consists of three layers, developed incrementally. The foundation layer provides the ability to complete a course in the absence of any obstacles. The second layer provides the ability to navigate around static obstacles. And the final layer provides the ability to work safely in traffic. The ability to follow a lane is the primary means of traveling on the road. Thus, the vehicle does not need very accurate or dense waypoints for navigation. The control system can track a path very accurately even at high speeds. The entire system integrates well to fulfill all the requirements of the UC.

7. References

Fox, D. (2003). Adapting the sample size in particle filters through KLD-sampling. *International Journal of Robotics Research*, 22:985-1004.

Fuerstenberg, K. C., Dietmayer, K. C. J. and Eisenlauer, S. (2002). Multilayer Laserscanner for Robust Object Tracking and Classification in Urban Traffic Scenes, In *Proceedings of the 9th World Congress on Intelligent Transport Systems*. Chicago, IL.

Fuerstenberg, K. C., Linzmeier, D. T. and Dietmayer, K. C. J. (2003). Pedestrian Recognition and Tracking of Vehicles using a Vehicle Based Multilayer Laserscanner. In *Proceedings of 10th World Congress on Intelligent Transport Systems*. Madrid, Spain, 2003.

Gat, E. (1998). Three-layered architectures. In D. Kortenkamp, R. P. Bonasso, and R. Murphy(Eds), *AI-based Mobile Robots: Case Studies of Successful Robot Systems*, p. 195-210, Cambridge, MA: MIT Press.

Gerkey, B., Vaughan, R. T., and Howard, A. (2003). The Player/Stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the 11th International Conference on Advanced Robotics (ICAR'03)*, pages 317-323, Coimbra, Portugal.

Guldner, J., Tan, H-S., and Patwardhan, S. (1997). Study of design directions for lateral vehicle control, In *Proceedings of the 36th Conference on Decision and Control*, San Diego, CA, pp. 4732-4737.

Hingwe, P. and Tomizuka, M. (1998). A variable look-ahead controller for lateral guidance of four wheeled vehicles, In *Proceedings of the American Control Conference*, Philadelphia, P, pp.31-35.

Lakhotia, A., Golconda, S., Maida, A, Mejia, P., Puntambekar, A., Seetharaman, G., Wilson, S. (2006). CajunBot: Architecture and algorithms. *Journal of Field Robotics*, 23(8):555-578.

Lu, F. and Milios, E. (1997). Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans. *Journal of Intelligent and Robotic Systems*, 18:249-275.

Maida, A. S., Golconda, S., Mejia, P., and Lakhotia, A. (2006). Subgoal-based local navigation and obstacle-avoidance using a grid-distance field. *International Journal of Vehicle Autonomous Systems*. 4(2-4):122-142

Pardo-Castellote, S. S. G. and Hamilton, M. (1999). NDDS: The real-time publish-subscribe middleware. Technical report, Real-Time Innovations, Inc.

Puntambekar, A. (2006). *Terrain mapping and obstacle detection for unmanned autonomous ground robots without sensor stabilization*, Lafayette, LA: University of Louisiana at Lafayette, The Center for Advanced Computer Studies, December, M.S. Thesis.

Shackleford, W. P., Proctor, F. M., and Michaloski, J. L. (2000). The neutral message language: A model and method for message passing in heterogeneous environments. In *Proceedings of the World Automation Conference*, Maui, Hawaii. http://www.isd.mel.ist.gov/documents/shackleford/Neutral_Message_Language.pdf.

Simmons, R. and James, D. (2001). IPC - A Reference Manual, version 3.6. Robotics Institute, Carnegie Mellon University. http://www.cs.cmu.edu/afs/cs/project/TCA/ftp/IPC_Manual.pdf.

Trepagnier, P. G., Nagel, J., Kinney, P.M., Koutsougeras, C. , and Dooner, M.(2006). KAT-5: Robust systems for autonomous vehicle navigation in challenging and unknown terrain, *Journal of Field Robotics*, 33(8):509-526, August.

van dan Elzen, C. (2004). Autovue Lane Departure Warning System, *Active Safety Systems for Heavy Trucks Workshop,* Federal Motor Carrier Safety Administration, March.

Vaughan, R. T. (2000). Gazebo: A multiple robot simulator. Technical Report IRIS-00- 394, Institute for Robotics and Intelligent Systems, School of Engineering, University of Southern California.

Venkitakrishnan, V. (2006). *CBWare - Distributed middleware for autonomous ground vehicles*, Lafayette, LA: University of Louisiana at Lafayette, The Center for Advanced Computer Studies, December, M.S. Thesis.

Wilson, Scott (2006). *A Real-time obstacle detection vision system for autonomous high speed robots*. Lafayette, LA: University of Louisiana at Lafayette, December, PhD Dissertation.